Departamento de Ingeniería Electrónica, de Sistemas Informáticos y Automática

INFORMÁTICA INDUSTRIAL II

PRÁCTICAS DE LABORATORIO

CURSO 2002-2003

PRÁCTICAS de INFORMÁTICA INDUSTRIAL II

ASISTENCIA

Las prácticas tendrán lugar en el Laboratorio de Arquitectura y Redes (Sótano, pasillo de la izquierda del edificio V.R.C) no siendo obligatoria la asistencia, permitiéndose la ausencia justificada hasta a un 20 % de las mismas, las cuales serán recuperadas con posterioridad.

Se entregarán todos los enunciados de prácticas al principio. El alumno deberá realizarlas a lo largo de las sesiones de prácticas. Una vez realizadas serán evaluadas por el profesor de prácticas, que resolverá cualquier duda presentada en la práctica. La práctica se debe traer preparada para su puesta en marcha en el laboratorio.

EVALUACIÓN

a) Las prácticas han de ser aptas para aprobar la asignatura.

Las prácticas de laboratorio se calificarán aptas, bien y muy bien en función de:

- Nivel técnico y calidad de redacción de la memoria de prácticas.
- Revisión del trabajo de prácticas funcionando por el profesor de prácticas.

b) Memoria de prácticas

Se entregará una memoria impresa de las prácticas realizadas, junto con un disco que contendrá los ficheros fuente de los diseños o programas si ha lugar. En la memoria debe aparecer claramente el nombre del alumno. Cada memoria de prácticas debe estar redactada mediante procesador de textos o equivalente y contendrá como mínimo los siguientes apartados:

1.- Portada donde aparezca claramente nombre, curso y fecha.
2.- Índice del documento.
3.- Un epígrafe para cada práctica, donde aparezca:

- Enunciado.
- Diseño del circuito con esquemas.
- Comentarios sobre el montaje y/o problemas presentados y como se han resuelto.
- Respuesta a las cuestiones planteadas en el enunciado.
- Posibilidades de ampliación/mejora si ha lugar.

4.- Anexos donde aparezcan fotocopias y/o documentación técnica si se considera necesario. Bibliografía que se ha empleado.

Todos los programas y diseños han de aparecer comentados, tanto los del disco como los de la memoria. Se valorará la claridad en este aspecto. Por imperativos legales la memoria no se devolverá al alumno.

MATERIAL USADO

Para la realización de las prácticas se hará uso del siguiente material:

1. Manuales específicos de circuitos integrados.
2. Equipo de desarrollo ALTAIR 537.
3. Entorno de desarrollo KEIL.
4. Monitor de depuración PAULMON2.

Manuales específicos de circuitos integrados:

Se trata de unos libros técnicos editados por los fabricantes, en los que podemos encontrar las características de los distintos circuitos integrados, tales como su función, su patillaje, su circuitería interna, su tabla de funcionamiento, sus características eléctricas y térmicas, etc.

Para la consulta en dichos manuales se dispone de varios índices:

- *Un índice alfanumérico*, donde se pueden encontrar los chips ordenados alfabéticamente según su nomenclatura y acompañados de la sección y página en la que se encuentran.
- *Un índice funcional*, en el cual los circuitos se ordenan según la función que realizan.

Para buscar un circuito integrado en un manual, la forma más sencilla consiste en buscar en el índice funcional hasta encontrar el chip adecuado a nuestras necesidades, averiguando su nombre. Una vez hecho esto nos iremos al índice alfanumérico, donde hallaremos la página en que se encuentra.

Equipo de desarrollo ALTAIR 537:

Es una placa equipada con el procesador 80C537 de Infineon, que no es más que un equivalente al 8051 con más periféricos. En el anexo se ofrecen los esquemas del circuito y toda la información auxiliar.

Entorno de desarrollo KEIL:

Integra un editor, ensamblador, compilador de C y simulador del 8051. Lo empleamos para el desarrollo de nuestros programas. En el anexo se acompaña un manual. Existen varias versiones, aunque todas ellas se basan en el mismo juego de directivas de compilación, solo cambia la facilidad de uso.

Monitor de depuración PAULMON2:

Consiste en un pequeño programa que se ejecuta en la placa ALTAIR 537 y nos permite descargar los programas realizados con el ensamblador o compilador en la placa y probar y depurar nuestros programas. Se acompaña un pequeño manual en el anexo.

PRÁCTICA Nº 1

ENTORNO DE DESARROLLO

OBJETIVOS

Introducir el manejo práctico en laboratorio del entorno de edición, ensamblado y depuración de programas KEIL. Introducir el uso del monitor de depuración PAULMON2.

CONCEPTOS BÁSICOS

Documentación teórica: Manual del compilador C51, Manual del entorno microvisión.

MATERIAL

PC con entorno KEIL
Placa de desarrollo ALTAIR 537

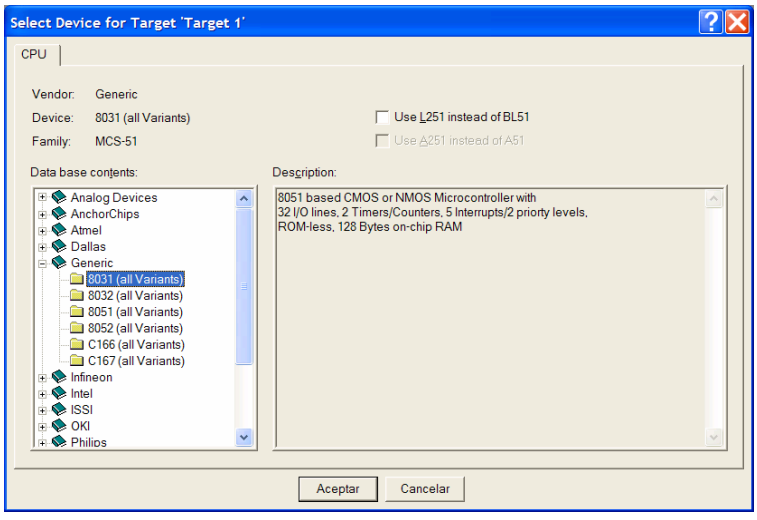
PROCEDIMIENTO

PARTE1 - Simulador

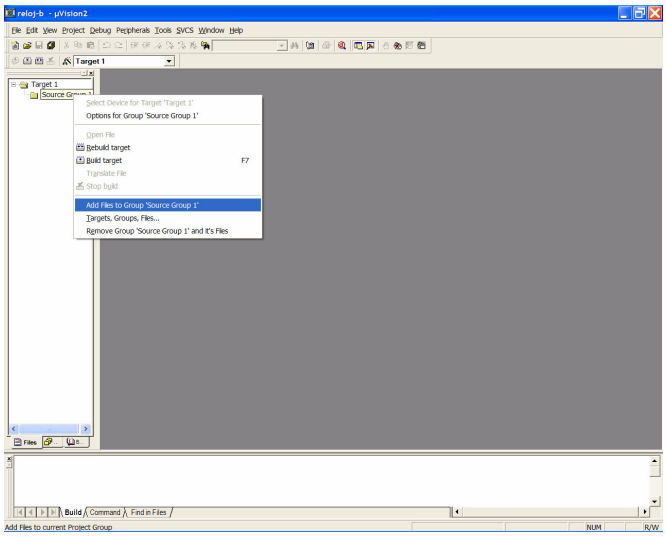
La primera práctica de laboratorio está diseñada para familiarizarnos con el lenguaje C, herramientas de software empleadas para escribir programas en C del 8051. Comenzaremos la práctica usando el software para introducir, ensamblar y simular un corto programa, que se da más abajo. Luego tendremos que escribir un corto programa y demostrar su funcionamiento empleando el simulador.

Se pretende realizar un reloj digital de forma que su salida se muestre en el dispositivo serie. El simulador dispone para ello de una ventana donde se realiza la simulación del puerto serie.

Lo primero será crear un proyecto nuevo. Seleccionaremos un dispositivo genérico 8031.

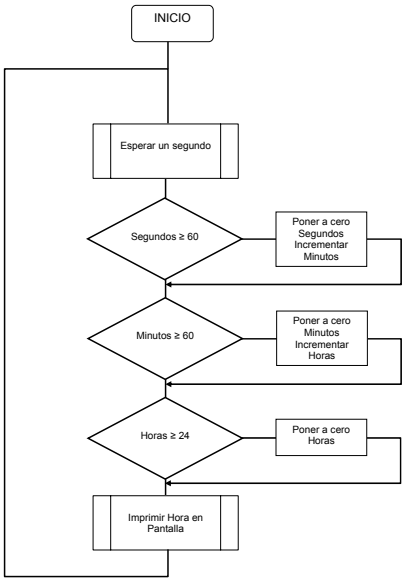


A continuación habra que redactar el programa e incluirlo en el proyecto seleccionando el árbol de la izquierda y con el botón derecho seleccionar Add Files to Group.



De esta forma el entorno quedará con el proyecto seleccionado, el fichero incluido en el árbol de compilación de la izquierda y listo para realizar la compilación y el linkado.

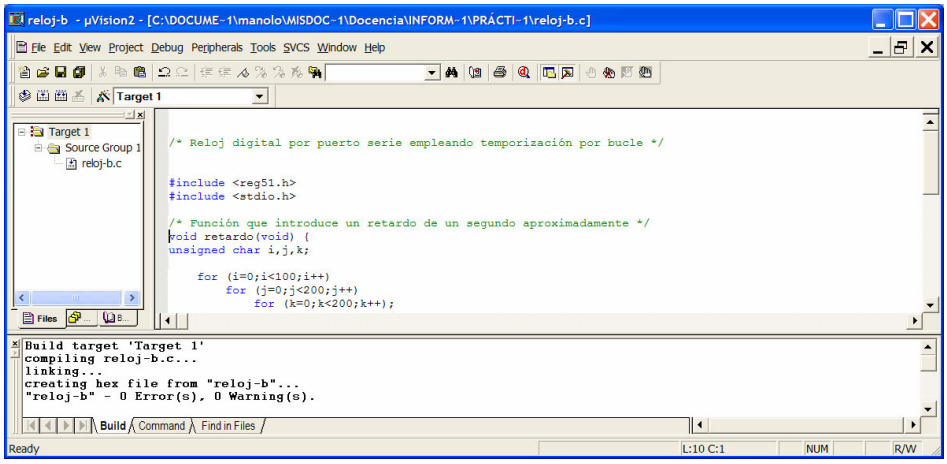
A continuación se muestra el diagrama de flujo y el código fuente:



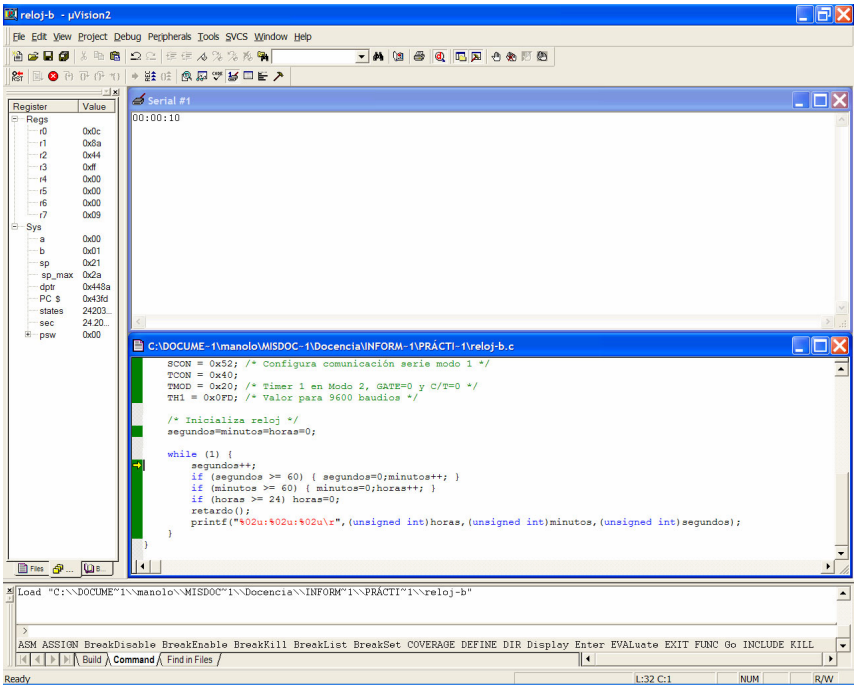

```
/*  Reloj  digital  por  puerto  serie  empleando
temporización por bucle */


#include <reg51.h>
#include <stdio.h>

/* Función que introduce un retardo de un segundo
aproximadamente */
void retardo(void) {
unsigned char i,j,k;

      for (i=0;i<100;i++)
            for (j=0;j<200;j++)
                  for (k=0;k<200;k++);
}

void main(void) {
unsigned char segundos,minutos,horas;

      /* Inicializa puerto serie */
      SCON = 0x52; /* Configura comunicación serie modo 1 */
      TCON = 0x40;
      TMOD = 0x20; /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
      TH1 = 0x0FD; /* Valor para 9600 baudios */

      /* Inicializa reloj */
      segundos=minutos=horas=0;

      while (1) {
            segundos++;
            if (segundos >= 60) { segundos=0;minutos++; }
            if (minutos >= 60) { minutos=0;horas++; }
```

```
            if (horas >= 24) horas=0;
            retardo();
            printf("%02u:%02u:%02u\r",(unsigned      int)horas,(unsigned
int)minutos,(unsigned int)segundos);
      }
}
```

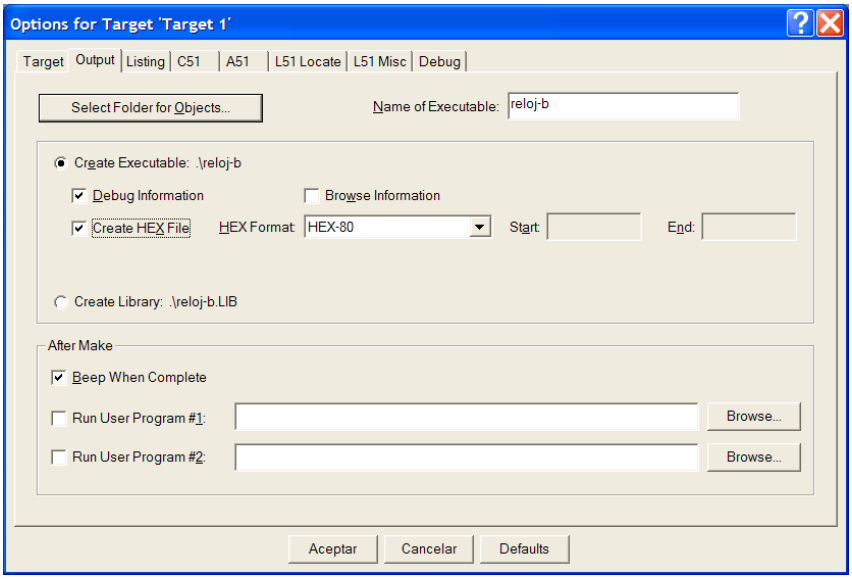
Introducir el código anterior empleando el editor del entorno KEIL, habiendo creado antes un proyecto.



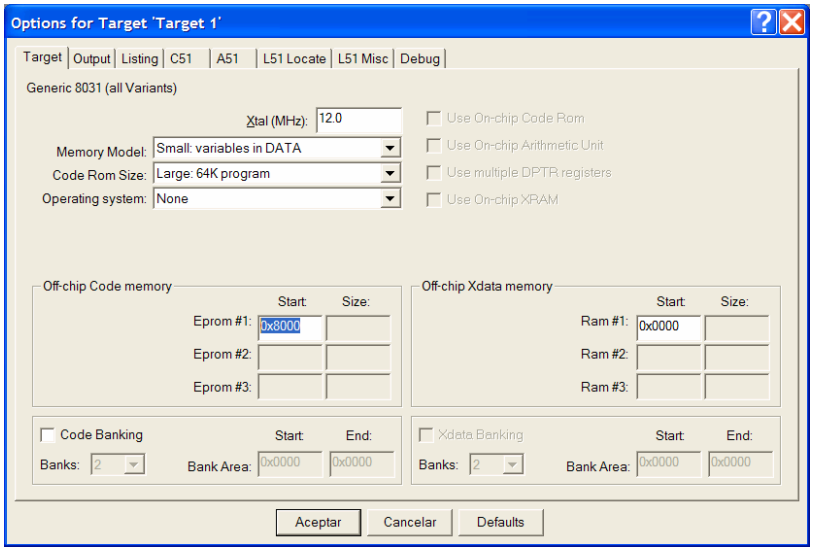
Compilar y linkar el código y finalmente emplear el simulador incluido en el entorno (d-Scope) para verificar que el código funciona.



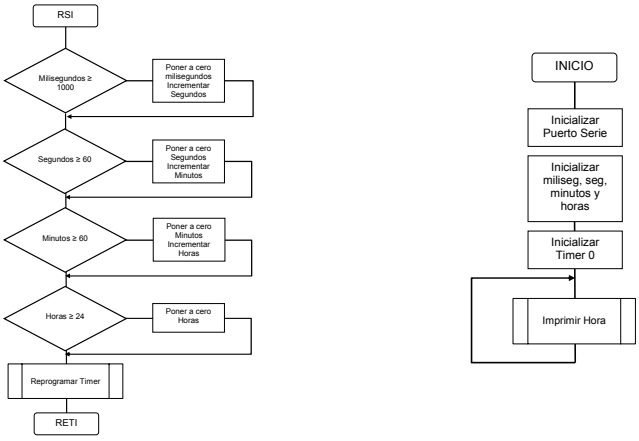
Para que el entorno genere un programa ejecutable (extensión .HEX) es necesario seleccionar la opción de creación de archivo HEX (Create HEX File).



Para probar el programa en la placa de desarrollo es necesario cambiar la memoria "off-chip" para que comience a partir de la dirección 0x8000 (32 K) y se pueda ejecutar. Esta opción no afecta al funcionamiento del simulador.



Realizar una versión diferente del programa basada en interrupciones. El diagrama de flujo será similar al siguiente:



El código fuente quedaría de la siguiente forma:

```
#include <reg51.h>
#include <stdio.h>
#define TIEMPO 1000

#define bajo(x)  ((unsigned char)((x)&0x00FF))
#define alto(x) ((unsigned char)((x)>>8))

unsigned char segundos,minutos,horas;
unsigned int msec;

/* Función que introduce un retardo de un segundo aproximadamente */
void timer0() interrupt 1 {

      msec++;
      if (msec>=1000) {msec=0;segundos++;}
      if (segundos >= 60) { segundos=0;minutos++; }
      if (minutos >= 60) { minutos=0;horas++; }
      if (horas >= 24) horas=0;
      /* Reprogramar el timer */
      TH0 = alto(65535-TIEMPO);
      TL0 = bajo(65535-TIEMPO);
}


void main(void) {
      /* Inicializa puerto serie */
      SCON = 0x52; /* Configura comunicación serie modo 1 */
      TCON = 0x40;
      TMOD = 0x20; /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
      TH1 = 0x0FD; /* Valor para 9600 baudios */

      /* Inicializa reloj */
      msec=segundos=minutos=horas=0;

      /* Inicializa timer 0 */

      TMOD |= 0x02;
      TH0 = alto(65535-TIEMPO);
      TL0 = bajo(65535-TIEMPO);
      TR0 = 1;  /* Pone en march timer 0 */
      ET0 = 1; /* Habilita int timer 0 */
      EA = 1; /* Habilitacion general de int */


      while (1) {
      printf("%02u:%02u:%02u\r",(unsigned           int)horas,(unsigned
int)minutos,(unsigned int)segundos);
      }
}
```

PARTE2 – Placa de desarrollo

Probar el programa anterior en la placa de desarrollo empleando el programa anterior en la placa de desarrollo. Descargar el programa a la placa y probar su correcto funcionamiento. El proceso es el siguiente:

- Paso 1. Eliminar del programa anterior las sentencias de inicialización del puerto serie, puesto que el PAULMON ya realiza inicializaciones del puerto serie y son innecesarias.

- Paso 2. Resetear la placa ALTAIR y pulsar la tecla <ENTER>. Esto provoca la aparición del mensaje de bienvenida del monitor PAULMON.

- Paso 3. Pulsar la tecla D (orden "Download"). De esta forma el programa monitor queda a la espera de recibir el programa que queremos prob

- Paso 4. Seleccionar del menú del programa la opción "Enviar archivo de Texto" y seleccionar el archivo HEX nuevo resultado de compilar y linkar la versión modificada del programa anterior.

- Paso 5. Cuando termina la transferencia el programa monitor muestra información estadística del número de bytes escritos en memoria. Nos sirve para confirmar que el programa se ha transferido con éxito.

- Paso 6. Pulsar la tecla G (orden "Go") para ejecutar el programa comenzando en la dirección 0x8000. El programa se ejecuta.

PRÁCTICA Nº 2

Manejo de un LCD. Empleo de librerías. Programa para adivinar un número.

OBJETIVOS

Emplear programación modular para el desarrollo de programas. Empleo de dispositivos periféricos.

MATERIAL

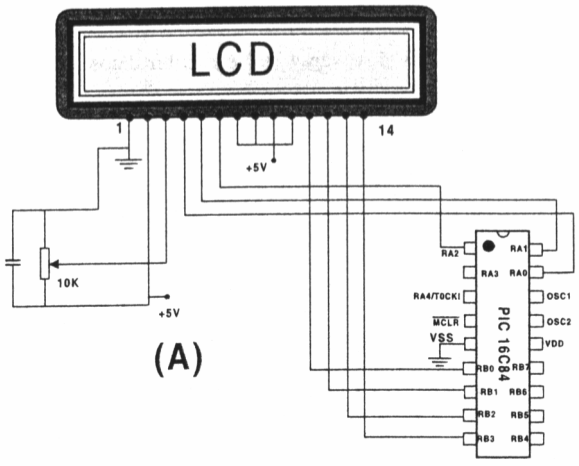
PC con entorno KEIL
Placa de desarrollo ALTAIR 537. Display LCD.

CONCEPTOS BÁSICOS

1.- Funcionamiento General.

El módulo LCD que vamos a trabajar tiene 14 patillas, cuya descripción se hace en la figura que sigue a este párrafo. Su alimentación es de +5 V, y la regulación del contraste se realiza dividiendo esos +5V mediante un potenciómetro de 10 k. Para el módulo de 8 bits requeriremos 11 líneas (uno de 4 bits necesitaría sólo 7). De ellas hay tres de control, que son EN (habilitación), I/D (Instrucción/Datos) y R/W (Lectura/Escritura).

En un modo de 4 bits usaríamos sólo las líneas DB4-DB7 de datos. El diagrama siguiente muestra su conexión a un microprocesador PIC, con el 8051, la conexión seria similar pero usando solo el puerto 1. El nibble bajo controlaría las patillas de control y el alto el nibble alto del LCD.



La activación de la línea EN (habilitación) es la que permite al LCD leer el resto de líneas, es decir, si la desactivamos no reaccionará ante los cambios en el resto de líneas. La línea R/W se conectará a masa, para ahorrar una línea, en todos los casos en los que no sea necesario el modo de lectura.

Pin	Nombre del pin	Función del pin
01	Vss	Masa
02	Vdd	+ 5 V
03	Vo ó Vee	Ajuste de contraste
04	I/D ó RS	Selección de modo
05	R/W	Lectura / Escritura
06	E ó EN	Validación (1) / Deshabilitación (0)
07	DB0	Línea de datos (bit de menos peso)
08	DB1	Línea de datos
09	DB2	Línea de datos
10	DB3	Línea de datos
11	DB4	Línea de datos
12	DB5	Línea de datos
13	DB6	Línea de datos
14	DB7	Línea de datos (bit de mas peso)

Habitualmente el puerto 1 del micro es utilizado para manejar las líneas de control (en la LCD.h P1.2 se conectará a EN, y habilitará la LCD, P1.1 manejará la lectura/escritura, y, finalmente, la P1.0 se encargará de la selección de modo), y también es utilizada para datos.

La secuencia de escritura debe seguir los siguientes pasos:

1) Línea I/D a 0 o a 1, según se trate de comandos o datos
2) Línea R/W a 0 (1 en caso de escritura)
3) Línea EN a 1 (se habilita la LCD)
4) Escritura de datos en el bus DB.
5) Línea EN a 0 (deshabilitación del LCD)

La misma secuencia en modo de 4 bits cambiaría:

1) Línea I/D a 0 o a 1, según se trate de comandos o datos
2) Línea R/W a 0 (1 en caso de escritura)
3) Línea EN a 1 (se habilita la LCD)
4) Escritura en los 4 bits de mayor peso del DB de la LCD.
5) Línea EN = 0
6) Línea EN = 1
7) Escribir de nuevo los 4 bits de menor peso
8) Línea EN = 0 (deshabilitación de la LCD).

Las dos secuencias de 4 bits se concatenarían dentro del LCD para formar 8 bits.

Al resetear un LCD o encenderlo éste se queda a la espera de instrucciones. Usualmente se suele empezar encendiendo la pantalla, colocando el cursor y configurando la escritura de derecha a izquierda.

La LCD contiene una RAM propia en la que almacena los datos, que se denomina DDRAM. Independientemente del número de caracteres visibles, la DDRAM contará con 80 posiciones. Los caracteres no visibles se visualizarán provocando un desplazamiento.

La utilización del LCD es lenta. Una escritura o lectura puede tardar entre 40 y 120 μsegundos; otras instrucciones pueden llegar a los 5 ms. Para lograr que el micro no necesite esperar tiene una instrucción de 1μseg que lee la dirección del contador y una bandera interior de ocupado. Cuando la bandera de ocupado (BF) está a 1, el LCD no puede leer ni escribir.

En nuestro ejemplo, del que a continuación mostramos el esquema, las líneas de datos se comparten con el teclado y una barra de diodos. Compartir la puerta B es una de las ventajas del PIC, puesto que le da una gran capacidad de reconfiguración, por su sencillez y rapidez.

1.1.- Habilitación de un LCD. Función lcd_habilita.

La línea EN de habilitación de una LCD necesita estar activada durante, al menos, 500 ns. La función lcd_habilita de LCD.C se asegura de que así sea, siendo LCDE = P1.2, es decir, EN:

```
// Envía un impulso de habilitación de 1us al LCD para
// completar la operación de escribir un registro o un carácter

sbit LCDE = P1^2;

void lcd_habilita(void) {

    ENA = 1;    // Pone a 1 la línea EN (habilita la LCD)
    _nop_();          // Pausa para 1us extra
    ENA = 0;    // Pone a 0 la línea EN (deshabilita la LCD)
}
```

1.2.- Selección de Modo (Comando/Datos).

La línea I/D selecciona entre el modo comando si vale 0 o el modo datos si es 1. Una llamada tipo lcd_comando( ) asegurará dicho modo antes de una habilitación de la LCD; lo mismo sucederá con lcd_carácter( ).

```
// Pasa el parametro constante software LCD de la tabla de
// igualdades. LCD_Comando saca el comando a la LCD y
// activa la línea de comando de la LCD, y la propia LCD
// mediante la llamada a LCD_Habilita, completando así
// el comando.

void lcd_comando(unsigned char comando) {
 lcd_chequea();      // Chequea la bandera de LCD ocupada
 RS = 0;             // Entra en modo registro
 PDIS |= (comando&0xF0);   // Primero se envia nibble alto
                     // de comando en nibble alto de puerto
 lcd_habilita();     // Envía el comando
 PDIS |= (comando&0x0F)<<4;// y luego nibble bajo de comando
                     // en nibble alto de puerto
 lcd_habilita();     // Envía el comando
```

```
}

// Pasa el caracter a imprimir en la LCD.
// Activa el modo datos y la propia LCD
// mediante la llamada a LCD_Habilita, completando así
// el comando.

void lcd_caracter(unsigned char carac) {

  lcd_chequea();      // Chequea la bandera de LCD ocupada
  RS = 1;                   // Entra en modo registro
  PDIS |= (carac&0xF0);     // Primero se envia nibble alto
                      // de comando en nibble alto de puerto
  lcd_habilita();           // Envía el comando
  PDIS |= (carac&0x0F)<<4;  // y luego nibble bajo de comando
                            // en nibble alto de puerto
  lcd_habilita();           // Envía el comando
}
```

Debemos recordar que es la línea R/W la que determina si se lee o escribe, debiendo estar debidamente activada según nuestros deseos antes de cualquier intento de acceso al LCD.

1.3.- Comando de la pantalla LCD.

Aunque pueden variar, en el caso que nos ocupa y en el estándar los comandos de la LCD son:

Comando	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Borra Pantalla	0	0	0	0	0	0	0	0	0	1
Cursor a Casa	0	0	0	0	0	0	0	0	1	*
Modo Introducción	0	0	0	0	0	0	0	1	I/D	S
Pantalla On/Off	0	0	0	0	0	0	1	D	C	B
Modo Desplaza-miento	0	0	0	0	0	1	S/C	R/L	*	*
Función	0	0	0	0	1	DL	líneas	Font	*	*
Dirección CGRAM	0	0	0	1	Dirección CGRAM					
Dirección DDRAM	0	0	1	Dirección DDRAM						
Lectura ocupado y dirección contador	0	1	BF	Dirección DDRAM						
Escribe RAM	1	0	Escribe Dato							
Lee RAM	1	1	Lee Dato							

Borra Pantalla: La borra y sitúa el cursor en su posición inicial (la 0).

Cursor a Casa: El cursor va a la posición inicial (la 0), pero sin borrar nada.

Modo instrucción: Configura la dirección del cursor I/D. Cuando I=1 incrementa la posición del cursor, y D=0 la decrementa. Mientras S=1 significa que hay desplazamiento en la pantalla. La operación se ejecuta durante la I/O de los datos.

Pantalla On/Off: Coloca en movimiento al cursor o genera desplazamiento en la pantalla. D para toda la pantalla, C para cursor On/Off, y B hace parpadear el cursor.

Desplazamiento Cursor/Pantalla: S/C indica el movimiento del cursor o desplazamiento en la pantalla, R/L la dirección a derecha o izquierda. No se varía el contenido de la DDRAM.

Función: DL indica la longitud de datos del interfaz; N el número de líneas de la pantalla y F el tipo de caracteres.

Dirección CGRAM: Coloca el dato enviado o recibido en la CGRAM después de este comando.

Dirección DDRAM: Coloca el dato enviado o recibido en la DDRAM después de la ejecución de este comando.

Bandera de ocupado BF: Lee BF indicando si hay una operación interna en curso y lee, además, el contenido de la dirección contador.

Escribe RAM: Escribe un dato en la RAM (ya sea DDRAM o CGRAM).

Lee RAM: Lee datos de la RAM (ya sea DDRAM o CGRAM).

Nombre Bit	Estado y Funcionamiento	
I/D	0 = Decrementa posición cursor	1 = Incrementa posición cursor
S	0 = Sin desplazamiento	1 = Con desplazamiento
D	0 = Pantalla Off	1 = Pantalla On
C	0 = Cursor Off	1 = Cursor On
B	0 = Parpadeo cursor Off	1 = Parpadeo cursor On
S/C	0 = Mueve el cursor	1 = Desplaza la pantalla
R/L	0 = Desplaza a la izquierda	1 = Desplaza a la derecha
DL	0 = Interfaz de 4 bits	1 = interfaz de 8 bits
Líneas	0 = 1 línea datos visible	1 = 2 líneas datos visibles
Font	0 = 5 x 7 pixeles	1 = 5 x 10 píxel
BF	0 = Puede aceptar instrucción	1 = Operación interna en curso

1.4.- Definición de valores de constantes de comandos para utilizar en las funciones de LCD.C.

Nombre constante	Valor	Significado
LCDLinea1	0x80	Coloca cursor en la posición 1 línea 1
LCDLinea2	0x0C	Coloca el cursor en la posición 1 línea 2
LCDCLR	0x01	Borra la pantalla + LCDLinea1
LCDCasa	0x02	Como LCDLinea1
LCDInc	0x06	El cursor incrementa su posición tras cada carácter
LCDDec	0x04	El cursor decrementa su posición tras cada carácter
LCDOn	0x0C	Enciende la pantalla
LCDOff	0x08	Apaga la pantalla
CursOn	0x0E	Enciende pantalla mas cursor
CursOff	0x0C	Apaga pantalla mas cursor
CursBlink	0x0F	Enciende la pantalla con cursor parpadeando
LCDIzda	0x18	Desplaza los caracteres mostrados a la izquierda
LCDDecha	0x1C	Desplaza los caracteres mostrados a la derecha
CursIzda	0x10	Mueve el cursor una posición a la izquierda
CursDecha	0x14	Mueve el cursor una posición a la derecha
LCDFuncion	0x38	Programa una interface 8 bits, pantalla 2 líneas, fuente 5x7 pixeles
LCDCGRAM	0x40	Programa el generador de caracteres del usuario RAM

1.5.- Direccionado de la RAM.

La RAM de una LCD no tiene direccionamiento continuo y lineal, pues el mapa depende de los caracteres y líneas que tenga el módulo.

Tamaño Pantalla	Visible	
Una Línea	**Posición Carácter**	**Dirección DDRAM**
1 x 8	00 – 07	0x00 – 0x07
1 x 16	00 – 15	0x00 – 0x0F
1 x 20	00 – 19	0x00 – 0x13
1 x 24	00 – 23	0x00 – 0x17
1 x 32	00 – 31	0x00 - 0x1F
1 x 40	00 – 39	0x00 - 0x27

Tamaño Pantalla	Visible	
Dos Línea	**Posición Carácter**	**Dirección DDRAM**
1 x 16	00 – 15	0x00 – 0x0F + 0x40 – 0x4F
1 x 20	00 – 19	0x00 – 0x13 + 0x40 – 0x53
1 x 24	00 – 23	0x00 – 0x17 + 0x40 – 0x57
1 x 32	00 – 31	0x00 - 0x1F + 0x40 – 0x5F
1 x 40	00 – 39	0x00 - 0x27 + 0x40 – 0x67

1.6.- Listado del módulo de control del LCD.

LCD.H:

```
/* Fichero include de lcd.c */

#ifdef _LCD_
/* Definición de todas las funciones, internas y exportadas */

void Pausa_5ms(void);
void lcd_init(void);
void lcd_caracter(unsigned char);
void lcd_comando(unsigned char);
void lcd_habilita(void);
void lcd_chequea(void);
void lcd_cString(unsigned char code *);
void lcd_dString(unsigned char data*);
void lcd_newline(void);
void lcd_borra(void);

#else
/* Definición de las funciones exportadas */

extern void lcd_init(void);
extern void lcd_cString(unsigned char code *);
extern void lcd_dString(unsigned char data *);
extern void lcd_newline(void);
extern void lcd_borra(void);
#endif
```

LCD.C:

```
      /*****************************************************************
      ; ***  LCD.LIB proporciona las siguientes funciones:              ***
      ; ***                                                            ***
      ; ***        - Configuración de las puertas                      ***
      ; ***        - Comandos en modo registro                         ***
      ; ***        - Exploración de LCD Busy - Ocupado                 ***
      ; ***        - LCD Enable (habilitación)                         ***
      ; *****************************************************************/

#define _LCD_
#include <reg51.h>
#include <intrins.h>
#include "lcd.h"

// Uso

      // Para iniciar la LCD después del encendido:
      // 1.- Llame lcd_init que inicializa el controlador de la LCD
      // Para escribir un comando o un carácter en la pantalla LCD:
      // 1.- Llamar lcd_comando ó lcd_caracter para enviar comando o carácter
      //     respectivamente a la pantalla LCD.

//****************** Comandos de Software para la LCD

#define LCDLinea1   0x80   // Dirección comienzo línea1
#define LCDLinea2   0x0C   // Dirección comienzo línea2
#define LCDCLR             0x01   // Borra pantalla, cursor a casa
#define LCDCasa            0x02   // Cursor a casa, DDRAM sin cambios
#define LCDInc             0x06   // Modo incrementa cursor
#define LCDDec             0x04   // Modo decrementa cursor
#define LCDOn       0x0C   // Pantalla On
#define LCDOff             0x08   // Pantalla Off
```

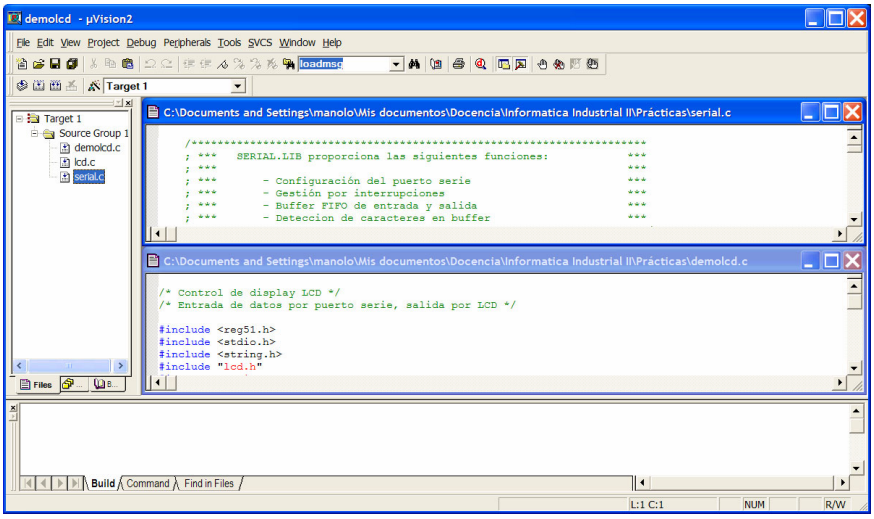
```
#define CursOn            0x0E   // Pantalla On, cursor On
#define CursOff           0x0C   // Pantalla On, cursor Off
#define CursBlink   0x0F  // Pantalla On, Cursor parpadeante
#define LCDIzda           0x10   // Mueve cursor a la izquierda
#define LCDDecha    0x14  // Mueve cursor a la derecha
#define LCDFuncion  0x38  // Inicializa registro función
#define LCDCGRAM    0x40  // Dirección origen CGRAM

/* Definicion de patillas de control */

#define PDIS P1           // 4 bits altos conectados a 4 bits altos del LCD
sbit ENA            = P1^2;             // Habilita LCD
sbit RS                   = P1^0;       // Selecciona Registro LCD
sbit RW             = P1^1;             // Lectura / Escritura LCD
sbit OCUPADO = P1^7;

unsigned char x=0,y=0;    /* Posición del cursor */

/* Definicion de funciones de control */

void   Pausa_5ms(void) {
       unsigned char i,j;

       for (i=0;i<10;i++)
              for (j=0;j<255;j++) _nop_();
       // Bucle de nops cerrado aprox 5ms
}

       /*****************************************************************
       ; *** Explora el estado de la bandera Busy (ocupado) de la LCD  ***
       ; *** Y espera que termine cualquier comando previo antes de    ***
       ; *** volver a la funcion que le llamó                          ***
       ; *****************************************************************/

void lcd_chequea(void) {
bit tmp;

       do {
              P1 |= 0x0F;  // Pone 4 bits bajos como entrada
              RS = 0;             // Coloca en modo registro
              RW = 1;             // Coloca el LCD en modo Lectura
              ENA = 1;     // Habilita el LCD
              Pausa_5ms(); // ...y espera
              tmp = OCUPADO;
              ENA = 0;
       } while (tmp);      // espera a que valga 1 (si es 0, está ocupado)
       RW = 0;             // Pone la LCD en modo escritura
}

// Envía un impulso de habilitación de 500 ns a la LCD para
// completar la operación de escribir un registro o un carácter
// La instrucción NOP sólo es necesaria para procesadores de
// una velocidad superior a 12 MHz. Si el procesador es de
// más de 16 MHz, se debe añadir un segundo NOP

void lcd_habilita(void) {

       ENA = 1;      // Pone a 1 la línea EN (habilita la LCD)
       _nop_();            // Pausa para 1us extra
       ENA = 0;      // Pone a 0 la línea EN (deshabilita la LCD)
}

// Pasa el parametro constante software LCD de la tabla de
// igualdades. LCD_Comando saca el comando a la LCD y
// activa la línea de comando de la LCD, y la propia LCD
// mediante la llamada a LCD_Habilita, completando así
// el comando.
```

```
void lcd_comando(unsigned char comando) {
	lcd_chequea();	// Chequea la bandera de LCD ocupada
	RS = 0;			// Entra en modo registro
	PDIS |= (comando&0xF0);	// Primero se envia nibble alto
		// de comando en nibble alto de puerto
	lcd_habilita();	// Envía el comando
	PDIS |= (comando&0x0F)<<4;// y luego nibble bajo de comando
		// en nibble alto de puerto
	lcd_habilita();	// Envía el comando
}

// Pasa el caracter a imprimir en la LCD.
// Activa el modo datos y la propia LCD
// mediante la llamada a LCD_Habilita, completando así
// el comando.

void lcd_caracter(unsigned char carac) {

	lcd_chequea();	// Chequea la bandera de LCD ocupada
	RS = 1;			// Entra en modo registro
	PDIS |= (carac&0xF0);	// Primero se envia nibble alto
		// de comando en nibble alto de puerto
	lcd_habilita();	// Envía el comando
	PDIS |= (carac&0x0F)<<4;  // y luego nibble bajo de comando
		// en nibble alto de puerto
	lcd_habilita();	// Envía el comando
}


	/*****************************************************************
	; *** Inicialización de LCD según el manual de datos de Optrex.   ***
	; *** Configura funciones de LCD para pantalla DMC16207           ***
	; *** Produce reset, borra la memoria y activa la pantalla.       ***
	; *****************************************************************/

void lcd_init(void) {
	lcd_comando(CursOn | LCDIzda);	// Da orden LCD y lo envia
	Pausa_5ms();			// ...y espera
	lcd_comando(0x28);		// Da orden inicia LCD y lo envia
	Pausa_5ms();			// ...y espera
	lcd_comando(0x28);		// Repite la operacion
	Pausa_5ms();			// ...y espera
/* Si deseas otra configuración de pantalla habrá de cambiar la siguiente
; lista de órdenes. Aquí encendemos la LCD, quitamos el cursor y borramos
; la pantalla, provocando que con cada carácter el cursor avance hacia la
; derecha. */
	lcd_comando(LCDInc);		// Modo Incremento
	lcd_comando(CursOff);		// Cursor off
	lcd_comando(LCDCLR);		// Borra pantalla
	lcd_chequea();			// Explora la bandera de ocupado
}

void lcd_cString(unsigned char code *cad) {

	while (*cad) {
		lcd_caracter(*cad);
		cad++;
	}
}

void lcd_dString(unsigned char data *cad) {

	while (*cad) {
		lcd_caracter(*cad);
		cad++;
	}
}
```

```
// Modificar para mas de dos lineas

void lcd_newline(void) {
       if (y==0)     {lcd_comando(LCDLinea2);y=1;}
       else {lcd_comando(LCDLinea1);y=0;}
}

void lcd_borra(void) {
       lcd_comando(LCDCLR);
       x=y=0;
}
```

2.- Uso de módulos: LCD y SERIAL.

Para incluir un módulo en nuestro proyecto tan solo habrá que incluir el fichero LCD.C en la lista de ficheros a compilar que aparece en el panel de la izquierda.



Deberemos incluir en nuestro fichero de programa principal la línea #include “lcd.h” y además #include “serial.h” para que aparezcan en el código fuente los prototipos de las funciones que nos ofrece el módulo. Estas funciones son las que podemos usar y de esta forma el compilador podrá calcular el espacio que necesita para almacenar los parámetros que se pasarán a las funciones del módulo auxiliar.

El uso de un módulo es bien sencillo si conocemos sus variables y sus rutinas internas. Aprovecharemos la creada en el apartado anterior para crear un pequeño programa que sitúe en la LCD la palabra 8051.

Pocas cosas debemos hacer resaltar para un paso tan sencillo, pero es importante saber que las funciones del fichero LCD.C se situarán delante de las del programa principal. No puede predecir, a priori, en qué dirección de memoria acabará cada módulo (el compilador lo detalla después de la compilación en el fichero pract3.LST), lo que si es evidente es que en la dirección de memoria 0000H el compilador coloca un salto al código presente en el fichero STARTUP.A51 y a continuación éste pasa el control a la función main().

2.1.- Listado del programa

LCDEMO.C:

```
/* Control de display LCD */
/* Salida de datos por puerto serie, salida por LCD */

#include <reg51.h>
#include <stdio.h>
#include <string.h>
#include "lcd.h"
#include "serial.h"

unsigned char data cadena[16];
unsigned char code txt1[]="Display Inic";

void main(void) {

       ser_init();               // Inicializa puerto serie
       lcd_init();               // Inicializa LCD

       while (1) {
              ser_cString(&txt1);
              lcd_cString(&txt1);       // Escribe cadena de texto EPROM
              ser_newline();
              lcd_newline();                    // Baja a la linea inferior
              strcpy(cadena,"Segunda Linea");  // Copia texto a RAM
              ser_dString(&cadena);
              lcd_dString(&cadena);     // Escribe cadena de texto RAM
              ser_newline();
              while(!keypress());       // Espera pulsacion de una tecla
              ser_getstring(&cadena);   //   Obtiene   todos   los   caracteres
entrados
              ser_dString(&cadena);     // Hace un eco
              lcd_borra();              // Borra display
       }
}
```

PROCEDIMIENTO

Realizar un programa que empleando la librería del puerto serie, y la librería del LCD inicialice el display y muestre por pantalla una cadena de caracteres que se ha pedido al usuario a través del puerto serie. Emplear para ello la siguiente librería del puerto serie, basada en el ejemplo de las clases de teoría:

SERIAL.H:

```
/* Fichero include de serial.c */

#ifdef _SERIAL_
/* Definición de todas las funciones */
unsigned char keypress(void);
void ser_init(void);
unsigned char ser_getchar(unsigned char);
unsigned char ser_putchar(unsigned char);
unsigned char ser_dString(unsigned char data *);
unsigned char ser_cString(unsigned char code *);
unsigned char ser_getstring(unsigned char data *);
unsigned char ser_newline(void);
#else
/* Declaración de funciones exportadas */
extern unsigned char keypress(void);
extern void ser_init(void);
extern unsigned char ser_getchar(unsigned char);
extern unsigned char ser_putchar(unsigned char);
extern unsigned char ser_dString(unsigned char data *);
extern unsigned char ser_cString(unsigned char code *);
extern unsigned char ser_getstring(unsigned char data *);
```

```
extern unsigned char ser_newline(void);
#endif
```

SERIAL.C:

```
      /*****************************************************************
      ; ***  SERIAL.LIB proporciona las siguientes funciones:          ***
      ; ***                                                           ***
      ; ***        - Configuración del puerto serie                   ***
      ; ***        - Gestión por interrupciones                       ***
      ; ***        - Buffer FIFO de entrada y salida                  ***
      ; ***        - Deteccion de caracteres en buffer                ***
      ; *****************************************************************/

#define _LCD_
#include <reg51.h>
#include "serial.h"

// Uso
// Para iniciar el puerto serie llamar ser_init que inicializa el puerto serie
// Para escribir un carácter en la consola llamar a ser_putchar o ser_cString
// para caracter o cadena respectivamente.
// Para leer emplear ser_getchar o ser_getString

#define uchar unsigned char
#define TRUE 1
#define FALSE 0

/* buffers que almacenan datos a transmitir
y datos recibidos en memoria externa de datos */
uchar xdata rbuf[32];
uchar xdata tbuf[32];

/* variables de manejo de fifos */
uchar rbin,rbout,tbin,tbout;
bit rfull,tempty,tdone;

/* Interrupción de puerto serie */
void serint(void) interrupt 4 using 1{
  /* si recibimos byte y no esta lleno el buffer
  lo introducimos en el buffer */
       if (RI && !rfull) {
              rbuf[rbin]=SBUF;
              RI=0;
              rbin=++rbin & 0x1F;
              /* si llegamos al final marcamos su llenado */
              /* otra funcion se encargara de procesar estos datos */
              if (rbin==rbout) rfull=1;
          }
       /* Si hemos terminado de transmitir byte
       transmitimos otro hasta tempty */
       else if (TI && !tempty) {
              SBUF=tbuf[tbout];
              TI=0;
              tbout=++tbout & 0x1F;
              /* Si ya no hay mas señalamos fin */
              /* otra funcion se encarga de colocar los
              nuevos datos a transmitir */
              if (tbout==tbin) tempty=1;
          }
       /* caso contrario ponemos a cero TI */
    else if (TI) {
              TI=0;
              tdone=1;
          }
}

void ser_init(void) {
```

```
	SCON = 0x60; /* Habilita int. serie y comunicación serie en modo 1 */
	TCON = 0x40;    /* TCON */
	TMOD = 0x20; /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
	TH1 = 0x0FD; /* Valor para 9600 baudios */
	IE = 0x90;
	REN = 1;

	tempty=tdone=1;
	rfull=0;
	rbout=tbin=tbout=0;
	rbin=1;
}

/* Lee una cadena de texto que haya sido entrada por
consola. El puntero sera como máximo 32 bytes. */

unsigned char ser_getstring(unsigned char data *cad) {
	if (((rbout+1)^rbin) != 0) {
		EA=0;
		while (((rbout+1)^rbin) != 0) {
			*cad=rbuf[rbout];
			rbout=++rbout & 0x1F;
			cad++;
			}
		*cad=0;		// Caracter de fin de cadena
		EA=1;
		return TRUE;
	}
	else return FALSE;
}

 /* Devuelve TRUE si hay caracteres en el buffer de entrada
	a la espera de ser leidos */

unsigned char keypress(void) {
	return (((rbout+1)^rbin) != 0);
}

	// Devuelve el siguiente caracter recibido y
	// cero si no se habia recibido ninguno

unsigned char ser_getchar(void) {
unsigned char cad;
	if (((rbout+1)^rbin) != 0) {
		EA=0;
		cad=rbuf[rbout];
		rbout=++rbout & 0x1F;
		EA=1;
		return cad;
		}
	return 0;
}

/* Escribe un caracter en consola
Devuelve TRUE si tuvo exito, FALSE sino lo tuvo */

unsigned char ser_putchar(unsigned char car) {
	if(((((tbin+1)^tbout) & 0x1F) != 0)) {
		EA=0;
		tbuf[tbin]=car;
		tbin=++tbin & 0x1F;
		if (tdone) {
				/* comenzar transmisión si todo acabo */
				tempty=0;
				tdone=0;
				TI=1;
			}
		EA=1;
```

```
            return TRUE; // Si se pudo mandar algo devolver cierto
      } else return FALSE;      // Sino devolver falso
}

/* Carga el buffer con un texto a transmitir
este esta situado en memoria de datos
Devuelve TRUE si tuvo exito, FALSE sino lo tuvo */

unsigned char ser_dString(unsigned char data *msgchar) {
      if((*msgchar != 0) &&
                  ((((tbin+1)^tbout) & 0x1F) != 0)) {
            EA=0;  // Desactiva Interrupciones
            while ((*msgchar != 0) &&
                        ((((tbin+1)^tbout) & 0x1F) != 0)) {
                  /* mientras haya caracteres y el buffer no se llene */
                              tbuf[tbin]=*msgchar;
                              msgchar++;
                              tbin=++tbin & 0x1F;
                              if (tdone) {
                              /* comenzar transmisión si todo acabo */
                                    tempty=0;
                                    tdone=0;
                                    TI=1;
                                    }
          }
            EA=1;  // Activa interrupciones
            return TRUE; // Si se pudo mandar algo devolver cierto
      } else return FALSE;      // Sino devolver falso
}

/* Carga el buffer con un texto a transmitir
este esta situado en memoria de codigo
Devuelve TRUE si tuvo exito, FALSE sino lo tuvo */

unsigned char ser_cString(unsigned char code *msgchar) {
      if((*msgchar != 0) &&
                  ((((tbin+1)^tbout) & 0x1F) != 0)) {
            EA=0;
            while ((*msgchar != 0) &&
                        ((((tbin+1)^tbout) & 0x1F) != 0)) {
                  /* mientras haya caracteres y el buffer no se llene */
                              tbuf[tbin]=*msgchar;
                              msgchar++;
                              tbin=++tbin & 0x1F;
                              if (tdone) {
                              /* comenzar transmisión si todo acabo */
                                    tempty=0;
                                    tdone=0;
                                    TI=1;
                                    }
          }
            EA=1;
            return TRUE; // Si se pudo mandar algo devolver cierto
      } else return FALSE;      // Sino devolver falso
}

/* Escribe salto de línea
Devuelve TRUE si tuvo exito, FALSE sino lo tuvo */

unsigned char ser_newline(void){
      return ser_putchar('\n');
}
```

Probar el modulo de comunicaciones serie. Probar otras variantes con el módulo LCD, como, por ejemplo, hacer parpadear el mensaje, o desplazarlo por la pantalla.

PRÁCTICA Nº 3

Empleo del BUS I2C para programación de EEPROM

OBJETIVOS

Manejo básico del bus I2C. Empleo de dispositivos periféricos.

MATERIAL

PC con entorno KEIL
Placa de desarrollo ALTAIR 537. Display LCD. Memoria EEPROM.

CONCEPTOS BÁSICOS

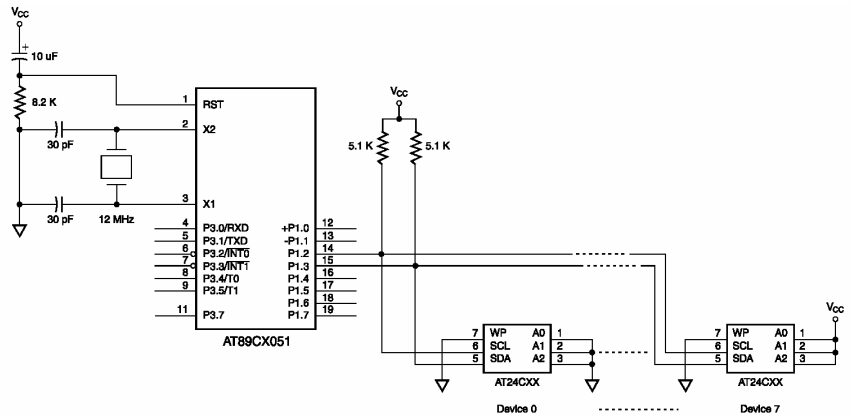
1.- Introducción.

Los dispositivos de memoria que usan bus serie ofrecen varias ventajas sobre los que emplean buses paralelos en aplicaciones donde se hacen aceptables velocidades de transferencia lentas. Además de necesitar menor espacio en la placa, los dispositivos serie permiten conservar patillas de E/S en las aplicaciones con microcontroladores. Esto se hace especialmente útil al añadir memoria externa a microcontroladores que no disponen de muchas patillas de E/S como sucede con el 87C751 de Phillips o el 89C2051 de Atmel.

A continuación mostramos una librería que emplea el bus I2C para leer o escribir los dispositivos EEPROM serie de la familia AT24CXX. El software se puede modificar para emplear otros dispositivos I2C diferentes.

2.- Hardware

Una interconexión típica entre un microcontrolador y una EEPROM serie de la familia AT24CXX aparece en la siguiente figura. Como se indica en la figura, hasta ocho miembros de la familia AT24CXX pueden compartir el bus, empleando las mismos dos patillas de E/S del microcontrolador. Cada dispositivo en el bus debe tener sus entradas de direccionamiento (A0, A1, A2) codificadas a una dirección única. En la figura, el primer dispositivo reconoce la dirección cero (A0, A1, A2 conectadas a nivel bajo), mientras que la octava reconoce la dirección siete (A0, A1, A2 conectadas a nivel alto).



No todos los miembros de la familia AT24CXX reconocen las tres entradas de dirección, limitando el número de estos dispositivos que pueden estar presentes a menos de siete. El número exacto de dispositivos de cada tipo que pueden compartir el bus se muestra en la siguiente tabla.

Device	Size (Bytes)	Page Size (Bytes)	Max Per Bus	Addresses Used
AT24C01	1K	4	1	None
AT24C01A	1K	8	8	A0, A1, A2
AT24C02	2K	8	8	A0, A1, A2
AT24C04	4K	16	4	A1, A2
AT24C08	8K	16	2	A2
AT24C16	16K	16	1	None
AT24C164	16K	16	8	A0, A1, A2
AT24C32	32K	32	8	A0, A1, A2
AT24C64	64K	32	8	A0, A1, A2
AT24C128	128K	64	4	A0, A1
AT24C256	256K	64	4	A0, A1
AT24C512	512K	128	4	A0, A1
AT24C1024	1M	256	2	A1

3.- Protocolo Bidireccional de Transferencia de Datos

El protocolo de transferencia bidireccional de datos empleado por la familia AT24CXX permite que un determinado número de dispositivos compatibles compartan un bus común de dos hilos. El bus consiste en una línea de reloj serie (SCL) y una línea de datos serie (SDA). El reloj se genera por el maestro del bus y los datos se transmiten en formato serie sobre la línea de datos, con el bit más significativo delante, sincronizados con el reloj. El protocolo soporta transferencias de datos bidireccionales en bytes de 8 bits.

Para esta aplicación, el microcontrolador sirve como el maestro del bus, iniciando todas las transferencias de datos y generando el reloj que regula el flujo de datos. Los dispositivos serie presentes en el bus se consideran esclavos, aceptando o enviando datos en respuesta a órdenes del maestro.

El maestro del bus inicia una transferencia de datos generando una condición de comienzo en el bus. Esto va seguido de la transmisión de un byte que contiene la dirección del dispositivo del destinatario que se pretende alcanzar. La dirección del dispositivo consiste en una porción fija de 4-bits y una porción programable de 3-bits. La porción fija debe coincidir con el valor previamente cableado en las patillas del esclavo, mientras que la porción programable permite al maestro seleccionar entre un máximo de ocho esclavos de tipo similar sobre el bus.

Las EEPROM serie de la familia AT24CXX responden a las direcciones de dispositivo con una parte fija igual a ‘1010’ y una parte programable coincidente con las entradas de dirección (A0, A1, A2). No todos los miembros de la familia AT24CXX examinan

las tres patillas de dirección. La tabla anterior muestra que dispositivos tienen en cuenta las patillas.

Los ocho bits de la dirección del dispositivo especifican una operación de lectura o escritura. Cuando el octavo bit se ha transmitido, el maestro deja flotante la línea de datos y genera un noveno pulso de reloj. Si un esclavo ha reconocido la dirección de dispositivo transmitida, responderá al noveno pulso de reloj generando una condición de reconocimiento en la línea de datos. Un esclavo ocupado cuando ha sido diseccionado puede no generar reconocimiento.

Esto sucede con la EEPROM AT24CXX cuando se esta realizando una operación de escritura. Tras la recepción del reconocimiento del direccionado del esclavo, el maestro continúa con la transferencia de datos. Si se ha ordenado una operación de escritura, el maestro transmite los datos restantes, con el reconocimiento enviado por el esclavo tras cada byte. Si el maestro ha ordenado una operación de lectura, deja la línea de datos flotante y genera pulsos de reloj para almacenar los datos enviados por el esclavo. Tras cada byte recibido, el maestro genera un reconocimiento en el bus.

El reconocimiento se omite tras la recepción del último byte. El maestro termina todas las operaciones generando una condición de parada en el bus. El maestro puede también abortar una transferencia de datos en cualquier momento generando una condición de stop.

Podéis revisar las hojas de características de la familia AT24CXX para obtener información más detallada sobre el funcionamiento del dispositivo y la temporización del protocolo de transferencia de datos sobre el bus.

Código fuente de la librería.

EEPROM.H

```
/* Fichero include de eeprom.c */

#ifdef _EEPROM_
/* Definición de todas las funciones */
bit lee_byte(unsigned char data *,unsigned int,unsigned char);
bit lee_actual(unsigned char data *, unsigned int);
bit escribe_byte(unsigned char data *,unsigned int,unsigned char);
bit start(void);
void stop(void);
bit shout(unsigned char);
unsigned char shin(void);
void ack(void);
void nak(void);
#else
/* Definición de las funciones exportadas por el modulo de lectura/escritura
de eeprom */
extern bit lee_byte(unsigned char data *,unsigned int,unsigned char);
extern bit lee_actual(unsigned char data *, unsigned int);
extern bit escribe_byte(unsigned char data *,unsigned int,unsigned char);
#endif
```

EEPROM.C

```
/* Conjunto de rutinas de manejo de memoria EEPROM */
/* emplear #include "eeprom.h"
```

```
/* para usar los encabezados en el programa */
/* (c) 2002 Informatica Industrial II EUPS La Rabida */

#define _EEPROM_
#include <reg51.h>
#include <intrins.h>
#include "eeprom.h"

#define FALSE 0     // Definicion de verdadero y falso
#define TRUE 1

/* Definición de las patillas de la interfaz */
/* Constantes de conexión de la EEPROM 24LC32 */

sbit SDA = P1^2;
sbit SCL = P1^3;

/*************************************************
; Función  LEE_BYTE (Lee un byte de la memoria EEPROM)
; Entrada:   puntero a byte, dir I2C.
; Salida:    byte leido, si error devuelve FALSE.
;*************************************************/

bit lee_byte(unsigned char data *byte,unsigned int dir,unsigned char dir_disp)
{
unsigned char tmp;

      /* Manda comando fantasma de escritura para fijar
         dirección interna */
      if (start()) {
            tmp=dir_disp;
            tmp <<= 1;          /* direcc. programable a 3:1 */
            tmp |= 0xA0; /* añade direccion fija */
            tmp &= 0xFE; /* especifica dir. escritura */
            if (!shout(tmp))    /* manda direccion de dispositivo */
                  return FALSE;        /* aborta si no ACK */
            if (!shout (dir>>4)) /* manda byte alto de direcciones */
                  return FALSE;        /* aborta si no ACK */
            if (!shout (dir&0x0F)) /* manda byte bajo de direcciones */
                  return FALSE;        /* aborta si no ACK */
            if (lee_actual(byte,dir_disp)) /* lee byte */
                  return FALSE; /* aborta si no ACK */
            stop();        /* parada del bus */
            return TRUE; /* lectura con exito */
      }
      else return FALSE; /* error de comienzo de transmisión */
}

/* Lee un byte de la dirección actual del AT24Cxx.
; Llamada con dirección programable dir. Dato en dato
; Retorna FALSE para indicar que el bus no está disponible
; o que el dispositivo direccionado no envia ACK. */

bit lee_actual(unsigned char data *dato, unsigned int dir) {
      unsigned char tmp;

      if (!start()) return FALSE; /* aborta si bus no disponible */
      tmp=dir;
      tmp <<= 1;          /* direcc. programable a 3:1 */
      tmp |= 0xA0; /* añade direccion fija */
      tmp |= 0x01; /* especifica dir. lectura */
      if (!shout(tmp))    /* manda direccion de dispositivo */
            return FALSE;        /* aborta si no ACK */
      dato=shin(); /*recibe byte de datos */
      nak();              /*no reconocer byte */
      stop();
      return TRUE; /*lectura con exito*/
}
```

```
/*************************************************
; Subrutina ESCRIBE_BYTE (Escribe un byte a la memoria EEPROM)
; Entrada:   DPTR direccion, A dato, B dir I2C.
; Salida:    si error devuelve FALSE, sino TRUE.
; Modifica:  Carry.
;*************************************************/

bit escribe_byte(unsigned char data *byte,unsigned int dir,unsigned char
dir_disp) {
unsigned char tmp;

      if (!start()) return FALSE; /* aborta si bus no disponible */
      tmp=dir_disp;
      tmp <<= 1;          /* direcc. programable a 3:1 */
      tmp |= 0xA0; /* añade direccion fija */
      tmp &= 0xFE; /* especifica operacion de escritura */
      if (!shout(tmp))    /* manda direccion de dispositivo */
            return FALSE;              /* aborta si no ACK */
      if (!shout(dir>>8)) /* Envia byte alto de direcciones */
            return FALSE;              /* aborta si no ACK */
      if (!shout(dir&0x0F))     /* Envia byte bajo de direcciones */
            return FALSE;              /* aborta si no ACK */
      if (!shout(*byte))  /* Envia datos a escribir */
            return FALSE;              /* aborta si no ACK */
      stop();
      return TRUE; /*escritura con exito*/
}


/* Manda START, definido como SDA alto a bajo con SCL alto
; Retorna con SCL, SDA a nivel bajo
; Retorna FALSE si el bus no está disponible */

bit start(void) {
      SDA = 1;
      SCL = 1;
      // Verifica que el bus este disponible
      if (SDA && SCL) {
      _nop_();
      SDA = 0;
      _nop_();
      _nop_();
      _nop_();
      _nop_();     // Cumple tiempo de Hold
      _nop_();
      SCL = 0;
      return TRUE;
      }
      else return FALSE;
}

/*  Manda STOP, definido como una transición bajo a alto en SDA
; junto con SCL a nivel alto.
; SCL se espera a nivel alto a la entrada.
; Retorna con SCL y SDA a nivel alto */
void stop(void) {
      SDA = 0;
      _nop_();     // Cumple SCL bajo y setup de datos
      _nop_();
      SCL = 1;
      _nop_();
      _nop_();     // Cumple retraso de setup
      _nop_();
      _nop_();
      _nop_();
      SDA = 1;
}
```

```
/* Desplaza un byte hacia el AT24CXX, el más significativo
; primero. SCL, SDA se esperan a nivel bajo a la entrada.
; Retorna con SCL bajo.
; Entrada: dato a transmitir
; Salida; FALSE para indicar fallo con el ACK del esclavo */

bit shout(unsigned char dato) {
unsigned char i;
bit tmp;

      i=8;
      while (i>0) {
            i--;
            SDA = (dato&(1<<i))? 1: 0;// Saca el bit
            _nop_();
            SCL = 1;     // Sube señal de reloj
            _nop_();     // Mantiene SCL alto
            _nop_();
            _nop_();
            _nop_();
            SCL = 0;     // Baja reloj
      }
      SDA = 1;     // Pone a 1 SDA para recibir ACK
      _nop_();     // Mantiene SDA bajo
      _nop_();
      _nop_();
      _nop_();
      tmp = SDA;   // Obtiene bit ACK
      SCL = 0;     // Baja reloj
      return tmp;
}

/* Desplaza un byte desde el AT24Cxx, el más significativo primero.
; Se espera SCL bajo a la entrada. Retorna con SCL bajo.
; Retorna dato byte recibido en A. */

unsigned char shin(void) {
unsigned char i,dato;
bit tmp;

      SDA = 1;     // hace SDA una entrada
      i=8;dato=0;
      while (i>0) {
            _nop_();     // Mantiene SCL bajo y setup de datos
            _nop_();
            _nop_();
            SCL = 1;     // Levanta reloj
            _nop_();     // Mantiene SCL alto
            _nop_();
            i--;
            tmp = SDA;   // bit de entrada
            dato |= ((unsigned char)tmp)<<i; // Guarda el bit
            SCL = 0;     // Baja reloj
      }
      return dato;
}

void ack(void) {
      SDA = 0;     // ACK bit
      _nop_();     // Mantiene SCL bajo y setup de datos
      _nop_();
      SCL = 1;     // sube reloj
      _nop_();     // Mantiene SCL alto
      _nop_();
      _nop_();
      _nop_();
      SCL = 0;     // Baja reloj
```

```
}

/* Saca un bit de reconocimiento negativo (alto)
; SCL se espera bajo a la entrada. Retorna con SCL bajo, SDA alto. */

void nak(void) {
       SDA = 1;      // NACK bit
       _nop_();      // Mantiene SCL bajo y setup de datos
       _nop_();
       SCL = 1;      // sube reloj
       _nop_();      // Mantiene SCL alto
       _nop_();
       _nop_();
       _nop_();
       SCL = 0;      // Baja reloj
}
```

PROCEDIMIENTO

Realizar un programa que empleando la librería del puerto serie, la librería del LCD y la librería de la EEPROM, pida al usuario que teclee una cadena de texto, la muestre por el LCD y la grabe en la EEPROM. Una vez grabada al comenzar el programa, este debe pedir al usuario si desea mostrar la cadena de texto grabada o bien introducir una nueva.